Anthony Rinaldi
Darin Beaudreau
Ethan Goldman

Adventurer's Archive

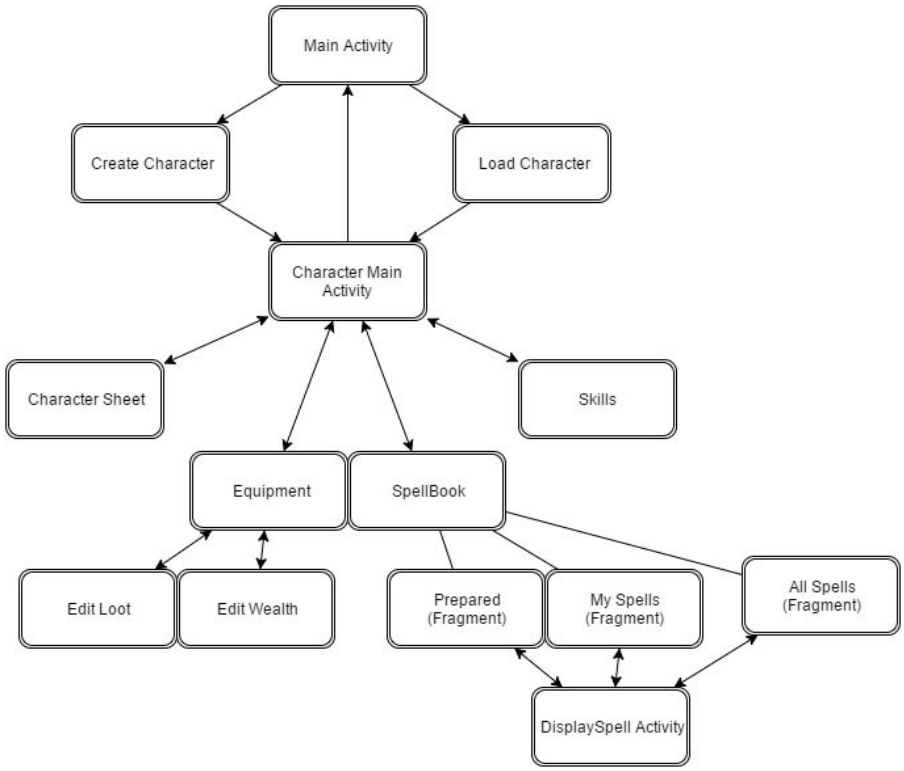
Project Goal

The purpose of this project was to create a handheld companion to the game Dungeons & Dragons. It is meant to keep track of all your characters items, skills, spells, and any other information that would pertain to a game of D&D.

Project Features

- Able to create and save character sheets.
    - Serializes the CharacterInfo class to write to saved files.
- Character sheets store important information about D&D 5th Edition characters.
- Character equipment and wealth is also stored.
- Reads spell information from a database so the user can add spells from a list to their spellbook for quick reference.
- Allows the player to roll skill checks via the Skills activity.
- The player can share their saved character sheets with their friends or the Dungeon Master, and they can then load that sheet on their device.

Project Design



File Structure

Darin Beaudreau (Project Leader, Programming)

- **MainActivity.java**, the logic behind the Main Activity. All this activity is for is launching the CreateCharacterActivity and LoadCharacterActivity. Basically just a “main menu”. Layout is **activity_main.xml**
- **CreateCharacterActivity.java**, the logic behind the character creation activity. It creates a new CharacterInfo object from the values in the various fields in the layout, which is then saved to a file using the name of the character. Once the character is created, it leads to the CharacterMainActivity. Layout is **activity_character_create.xml**. Leads to MainActivity if back/up are pressed.
- **LoadCharacterActivity.java**, the logic behind the layout in **activity_character_load.xml**. This activity lists the saved character files found in the app directory. There are buttons to either Load or Delete the character. Delete does exactly what it says, and removes that character from the list. Load will set the global CharacterInfo object to the deserialized object in the character file and lead to CharacterMainActivity. Leads to MainActivity if back/up are pressed.
- **CharacterMainActivity.java**, the logic behind **activity_character_main.xml**. This activity serves as a “hub” to the various activities that provide information about the character. It has buttons that lead to the CharacterSheetActivity, CharacterEquipmentActivity, CharacterSpellbookActivity, and CharacterSkillsActivity activities. Leads back to MainActivity when back is pressed.

- **CharacterSheetActivity.java**, the logic behind **activity_character_sheet.xml**. This file is massive, as I had to provide listeners for a lot of fields that, when changed, update other fields. This activity took the longest out of any activity in the entire app, as it was a lot of information and logic to set up. Leads back to CharacterMainActivity when back/up are pressed.
- **CharacterSpellbookActivity.java**, sets up the TabLayout in **activity_character_spellbook.xml**, which contains the PreparedSpellsFragment, MySpellbookFragment, and FullSpellbookFragment fragments. Also establishes the database so that the fragments can query it. If the database does not exist or is empty, it creates it and fills it with data from the phb_spells.xml file by parsing it and inserting it into the database. Leads back to CharacterMainActivity when back/up are pressed.
- **PreparedSpellsFragment.java**, has an ExpandableListView that displays spells that the user has prepared for the day. Spells can be prepared from the DisplaySpellActivity, and can only be prepared if they are in the character's spellbook.
- **MySpellbookFragment.java**, has an ExpandableListView that displays spells that the user has in their spellbook. Spells can be added to the spellbook by going to the All Spells tab and selecting a spell, the tapping "Add/Remove" in the action overflow menu.
- **FullSpellbookFragment.java**, has an ExpandableListView that populates with all spells in the game, filtered by the character's class.
- **SpellbookTabAdapter.java**, an adapter for the TabLayout in CharacterSpellbookActivity. It returns the proper fragment based on which tab the user scrolls to.
- **DisplaySpellActivity.java**, the logic behind **activity_display_spell.xml**. This activity queries the database using the spell's name passed in the intent bundle for all stored information on that spell and then displays it using the fields in the layout file. Leads back to CharacterSpellbookActivity when back/up are pressed. I could not figure out a way to return to the same Fragment that was open when the activity was launched, as based on what is done in this activity, the list in those fragments must be updated.
- **SpellListAdapter.java**, handles displaying the spells in the ExpandableListView in each of the fragments.
- **GlobalState.java**, extends Application and contains the CharacterInfo object used to retrieve information from the character object throughout the various activities. Also contains static methods for saving and loading the character. Also contains the database object so the various activities can query it by accessing the application context.
- **CharacterInfo.java**, the character object, which stores all information about the character, including all fields from the character sheet, the Equipment objects, lists of integers of spell ids to be used when querying the database, and other fields.
- **AbilityScore.java**, contains an enumerator for the 6 ability scores and several member variables for storing information about those ability scores.
- **CharacterRace.java**, contains an enumerator for all of the character races defined in the Player's Handbook. Has member variables and methods to retrieve information about the character's race.

- **CharacterClass.java**, contains an enumerator for all of the character classes defined in the Player's Handbook. Has member variables and methods to retrieve information about the character's class.
- **CharacterAlignment.java**, contains an enumerator for the 9 alignments on the spectrum along with member variables and methods for getting information about them.
- **Skill.java**, contains an enumerator about the 18 character skills in the game, as well as various helper methods and member variables for getting information about them.
- **Spell.java**, just a container class to store various strings and integers pertaining to information about spells.
- **SpellXMLParser.java**, contains a static method to parse phb_spells.xml and create Spell objects, which can then be inserted into the database. Only to be used if the database is not present, or is empty.
- **DBHelper.java**, extends SQLiteOpenHelper, and allows the user to create, and then read/write from/to the database. Mostly uses raw queries.
- **DiceRoller.java**, contains static helper methods for generating random die rolls via Java's Random class. This is used anywhere in the app that requires the result of a dice roll. Supports any number of sides.
- Also created the XML files listed here:
    - character_list_item.xml
    - spell_list.xml
    - spell_list_group_item.xml
    - spell_list_item.xml
    - items.xml
    - spell_details_menu.xml
    - phb_spells.xml (found on Google)

Ethan Goldman (Programming, Project Idea)

- **CharacterEquipmentActivity.java**, and its layout file **activity_character_equipment.xml**. Displays the player's wealth and a list of the player's equipment. There are buttons to add equipment and add wealth. Also displays the player's encumbrance level and total weight. Leads to CharacterMainActivity when back/up are pressed.
- **EquipmentDetailsActivity.java**, and its layout file **activity_equipment_details.xml**. Used to create new Equipment objects to add to the list of Equipment in CharacterEquipmentActivity, or to delete Equipment from the list. Has several fields for the player to fill out. Leads to CharacterEquipmentActivity when back/up are pressed.
- **UpdateCashActivity.java**, and its layout file **activity_update_cash.xml**. Used to add/subtract money from the player's total wealth. Stores the information in a Coins object in the CharacterInfo class, which uses the conversion rates specified in the Player's Handbook to convert different types of coins. For example: 10 gold = 1 platinum. So if the player says that they have 20 gold, the activity will convert that to 2 platinum, and so on.

- **Equipment.java**, used to store information about the Equipment that is displayed in the Equipment list in CharacterEquipmentActivity. The value of the Equipment is defined in a Coins object.
- **Coins.java**, a class used to represent a monetary value. Contains methods for converting coin values to others based on conversion rates given in the Player's Handbook. Money is represented by a number of Copper, Silver, Electrum, Gold and Platinum coins.
- Also created **equipment_list_item.xml**

Anthony Rinaldi (Programming, Team Report)

- **CharacterSkillsActivity.java**, and its layout file **activity_character_skills.xml**. This activity lists the 18 skills in the game. It allows the user to specify which skills they are trained in via a CheckBox, and allows them to make skill checks via a "Roll" button to the right of each skill. This roll button creates a Toast that displays the result of the roll broken down into the die roll, the bonus for the skill, and the total.